Python 包应用程序编程接口破坏性变更检测系统

杜星亮[1)] 马 骏[1)]

[1)](南京大学，计算机科学与技术系，计算机软件新技术国家重点实验室)

摘 要 随着近年来 Python 编程语言的流行，社区开发者创建并维护了大量的第三方软件包。在包的演化过程中，应用程序接口（API）频繁发生变更，开发人员需要保持版本的向后兼容性来避免破坏下游代码。大量的开发者和用户，以及 Python 的动态特性和灵活设计，使得检测 Python 包的破坏性变更变得重要且富有挑战性。尽管 Python 十分流行，目前仍缺少较好解决这一问题的自动化工具。我们在 ISSRE'22 会议接收的论文中，提出了基于 Python 包 API 模型的系统化检测方法，使用程序分析提取 API 并使用约束检测分析变更。我们实现了包括预处理，API 提取，变更检测和破坏性分析四个步骤的原型系统 AexPy，关注健壮性和扩展性，分解处理步骤并使用缓存和增量分析提高效率，支持容器部署，提供了浏览器和命令行接口。在真实 Python 包的评估实验中，AexPy 在 61 个已知破坏性变更数据集上达到了 86.9% 的召回率，并发现了相关软件包最新版本中未被记录的破坏性变更，已有 31 个新发现变更得到了开发者的确认，具有一定的实用价值和推广前景。

关键词 Python，应用程序编程接口，向后兼容性，破坏性变更

1 引言

Python 便捷的标准库和庞大的活跃第三方软件包生态帮助开发者高效地实现想法，吸引了大量开发者。在包的演化过程中，可能因为错误修复或新功能引入而重新设计数据结构，改变数据操作或调整执行约束。其中一些变化可能会破坏下游代码，即所谓的“破坏性变更”[1]。一个常见的情况是应用程序编程接口（API）变更。API 描述了一个包的公共接口，这些接口是下游代码所依赖的契约，所以 API 变更很可能会影响下游代码。在流行 Python 框架包的演化中，超过 40% 的 API 变更是破坏性的，这一比例高于传统静态语言。这些破坏性变更破坏了向后兼容性，降低了演化的稳定性，并可能在下游代码中引起潜在的错误。

由于 Python 的动态特性和灵活设计，检测这些包演化中的破坏性变更是有挑战性的，如动态模块导入，动态类型，复杂的 API 引用关系，灵活参数传递等，与传统语言相比，这些挑战都增加了 API 的复杂性，引入了更多种类的变更，提高了分析难度。现有的方法缺乏对这些挑战的充分考虑，导致了不精确和不完整的分析结果。

我们在 ISSRE'22 会议论文[2] 中提出了一种系统化方法来自动检测 Python 包 API 破坏性变更，这有助于提高 Python 包演化的稳定性和可靠性。我们的方法主要分三个步骤：从不同版本的 Python 包中提取 API，比较 API 以检测不同模式的变更，并评估变更的向后兼容性，以确定破坏性严重程度。

我们设计和实现的原型系统 AexPy 使用程序分析和约束检测等技术，结合 Python 标准库和 Mypy 类型检查工具[3]，完整实现了论文提出的方法。得益于细致的模型和混合分析方法，AexPy 在 61 个已知变更中达到了 86.9% 的召回率，较现有工具提升近 50%，且具有同一数量级的耗时。AexPy 也检出了真实软件包的潜在破坏性变更，并得到了开发者的确认，具有一定的实用价值和推广前景。

AexPy 系统设计强调健壮性和扩展性，并关注分析效率和结果复用。在算法实现中，采用分步隔离，增量计算的原则，将分析过程划分为预处理，API 提取，变更检测和破坏性评估四个步骤，通过抽象和复用中间结果，增量分析，保证了实现的健壮性和效率。在系统架构设计中，采用职责解耦，接口统一的原则，解耦输入输出，缓存机制，和算法执行，使系统易于维护和扩展。AexPy 使用容器镜像打包，易于重现和迁移，并提供了命令行，RESTful API 和独立的前端界面，便于开发人员使用和自动化集成部署。

本文如下组织后续章节。第2节介绍了 Python 包 API 破坏性变更检测问题的背景和面临的挑战。第3节给出了 AexPy 背后的方法原理。第4节描述了系统架构设计和核心实现思路，并简要说明了实验评估结果。AexPy 的完整方法原理和实验结果可参见我们的论文[2]。第5节介绍了如何获取和使用 AexPy 。第6节对本文进行了总结。

杜星亮，E-mail: xingliangdu@smail.nju.edu.cn. 马骏，E-mail: majun@nju.edu.cn.

2 问题背景

软件包的 API 描述并规定了“预期行为”，包含数据的描述和操作[4]。向后兼容性指包的下游能够以与旧版本相同的方式使用新版软件包并获得相同的行为，即具有语法和语义上兼容的 API[1]。以不遵守向后兼容的方式修改包被称为破坏性变更，可能导致下游代码发生编译时或运行时错误。

Python 是一种解释型，动态类型，多范型的编程语言，长期被列为最受欢迎的编程语言之一。官方仓库 Python 包索引（PyPI）截至 2022 年 10 月，包含了超过 40 万个包。Python 包中的破坏性变更频繁发生并产生了广泛的影响。张 等人 [5] 调研了六个 Python 框架和其下游项目，发现与 Java 项目相比，Python 包更频繁地发生破坏性变更，并影响了超过半数的下游项目。

不同于传统编程语言，Python 包中的破坏性变更检测问题具有自身的特点和挑战性。这主要来自 Python 的动态特性和灵活设计，而现有工具和方法缺少对这些挑战的充分考虑。其一是**动态语言特性**。作为动态语言，程序行为主要发生在运行时，语言也提供了装饰器等机制来动态修改程序行为。Python 缺少编译时的类型声明和检查，其支持鸭子类型的动态类型系统带来了复杂多样的类型变更。类型标注有助于获取类型信息，但类型变更的检测仍有困难。其二是**复杂的 API 引用关系**。由于 Python 允许用不同的名字来引用同一个 API，所以很难定位和区分 API。这一特性在一些流行包中被广泛用来创建短名称，但也导致了复杂交错的 API 组织结构。第三是**模糊的 API 可见性**。Python 没有访问修饰符，这导致了 API 可见性的混乱。开发者通常有自己的私有命名约定，但由于缺少对应语言机制，下游代码仍然可以轻松访问这些成员。同时，同一 API 的不同别名可能指示了不同的可见性。现有工具忽视了这些伪私有 API，导致遗漏了相关破坏性变更。其四是**灵活的参数传递**。除了可选参数和默认值，Python 还支持多种传参方式，包括位置参数，关键字参数，和带名称或不带名称的变长参数列表，这带来了复杂的签名变更情况。

3 方法原理

3.1 API 建模与提取

根据 Python 项目的模块化结构，我们将软件包 API 集合 E 建模为四类 API 集合的并集：模块 M，类 C，函数 F，和属性 A。对于**模块**，使用成员名到成员 API 对象的映射来建模成员关系。对于**类**，考虑其基类列表，抽象基类列表，和方法解析顺序来建模继承关系。对于**函数**，考虑包括参数和返回类型的签名信息，并按所属范围划分为静态、类、和实例方法。对于函数中的参数，关注名称、位置、可选性、默认值、传参方式、和类型。对于**属性**，考虑其类型，并按所属范围划分为静态（模块）、类、和实例属性。特别地，我们设计了一个简单可移植，不依赖于具体类型检查器的类型系统来表示 API 中的类型信息。这一系统包含三类原子类型（字面值、类类型、特殊类型）和由原子类型组合而成的四类复合类型（和类型、积类型、调用类型、泛型类型）。为建模 API 引用关系，我们使用 API 定义位置的完全限定名作为其标识符，并将所有其他指向这一 API 的名称记为别名。

基于这一 API 模型，AexPy 结合动态反射和静态分析来应对之前提到的挑战。首先，通过反射处理由动态特性导致的动态行为：在虚拟环境中安装软件包后，导入所有模块，并使用自顶向下的广度优先搜索发现包中能够运行时访问到的所有 API，并使用标准库`inspect`模块收集 API 元数据。由于被不同名称引用的同一 API 在运行时是同一对象，所以这一过程收集到了准确的 API 别名信息。反射无法获取类型标注信息和函数体内的 API 信息，如实例属性等，故根据 API 定义位置将 API 对象与其源代码片段对应后，引入静态分析增强 API 信息：通过遍历构造函数语法树中的所有赋值语句发现实例属性；使用静态分析工具 Mypy[3] 分析代码类型信息，从 Mypy 内部生成的类型表示中，提取对应于每个 API 元素的类型信息，并映射到我们设计的类型系统中。

3.2 变更分类与检测

根据我们提出的 API 模型，我们将 API 变更按发生位置（模块，类，函数，属性，参数，别名）和变更形式（增加，移除，修改）划分为 17 种粗粒度变更模式，并进一步细化为 42 种变更模式，包括增加模块、移除基类、移动参数、修改返回类型、移除外部别名等[6]。

我们设计了一个差异比对算法来检测这 42 种变更模式。AexPy 首先配对两个版本的 API 集合对应元素，然后使用约束检测判断是否发生各类模式的变更。在配对阶段，首先根据 API 完全限定名和别名，模拟 Python 名称解析过程，配对每个 API

名称在两个版本中解析到的实际 API 元素，进一步地，对于函数参数，区分不同传参类型，配对在合法传参方式下接受同一实参的参数。对于无法配对的 API 元素，使用 ⊥ 表示对应 API 缺失。在检测阶段，我们为每个变更模式关联了一个逻辑表达式作为判断变更是否发生的约束条件，包括通过判断配对是否包含 ⊥ 来检测 API 增加或移除，和比较 API 信息来检测 API 修改。

3.3 破坏性分级与评估

根据对开源仓库和问题报告的观察，我们引入了启发性破坏性级别来指示变更影响范围，即破坏性严重程度。由于 Python 对 API 声明的弱约束和有限的编译期检查，开发者难以限制下游代码对 API 的使用方式，从而导致复杂的变更影响情况。细粒度的破坏性级别有助于开发者在获得完整变更信息的同时，关注最为重要的变更，从而进一步提高实用性。除兼容性变更外，有以下三个级别的破坏性变更。AexPy 根据广泛使用的命名约定分析 API 别名来区分伪私有和公开 API，并根据变更模式和内容为变更分级。对于类型变更，AexPy 使用我们针对类型系统中每类类型设计的兼容性推导规则判断变更前后的类型是否兼容。

低 发生在伪私有 API 和外部别名上的变更。下游尽管能够访问，但通常不会直接使用这些 API。

中 不直接破坏 API 可用性的变更，指示破坏性场景是有条件的或稀少的。如由于鸭子类型，参数类型变更只会在函数执行逻辑中实际访问或检查参数时报错，而不会在传参调用时报错。

高 直接破坏 API 可用性的变更。如移除类或函数，添加必需参数等，这会导致访问相关 API 或函数传参时立即报错。

4 系统实现

4.1 架构设计

为提高系统的扩展性，支持命令行，Web API，前端界面等多种输入输出交互方式，并通过缓存机制最大化利用分析结果，提高分析效率。AexPy 的系统设计遵循**职责解耦，接口统一**的原则，并采用分层结构，下层向上层提供服务，直至面向用户的最上层。如图 1所示，算法模块尽可能将分析算法实现为不含副作用，可重入的纯函数，不考虑模块依赖和数据存取。缓存模块负责将算法模块的分析结果存储到文件系统或文档数据库中。分析和缓存管理器层将不同算法和缓存实现抽象为同一接口，封装实现细节。服务提供层根据环境配置构建分析器和缓存管理器，并为每一分析阶段，实现缓存读取、算法执行、缓存存储的完整运行流程，合并缓存和分析接口。流水线层根据每个分析阶段指定的分析算法，将阶段间输入输出组合为流水线，提供高层次接口供上层使用。最后，用户界面层将用户输入分发到对应流水线进行处理，完成整个流程。

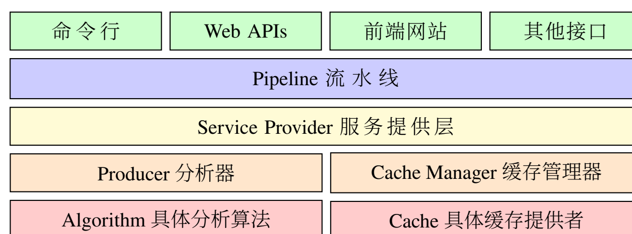

图 1 AexPy 系统架构

4.2 算法实现

为提高健壮性和效率，AexPy 的分析算法实现遵循**分步隔离，增量计算**的原则，将分析过程拆分为四个阶段，为每个阶段间的中间结果设计了数据结构，引入缓存机制保存重用中间结果，并根据分析方法层层递进、逐步深化的特点，使用增量计算提高效率。如图 2所示，AexPy 首先对指定版本的包进行预处理，获得包源码和环境依赖信息；然后根据环境信息构建匹配的虚拟环境，安装软件包并使用前述方法提取 API；得到两个版本的 API 描述后，使用前述匹配和约束检测算法检测 API 变更；最后根据破坏性分级规则，评估每个变更的破坏性级别，并生成报告。

4.3 效果评估

我们从 GitHub 上近一年的已解决 Python 包破坏性变更问题报告和流行包的变更日志中，收集了 61 个已知破坏性变更和相关的 45 个软件包，并与两个现有工具 PyCompat[5] 和 Pidiff[7] 进行了比较。

召回率上，在 61 个已知破坏性变更中，AexPy 检出了 53 个变更，召回率 86.9%，较 PyCompat（检出 22），Pidiff（检出 23），提高了近 50% 的召回率。PyCompat 有 13 个变更报错，Pidiff 有 31 个，与之相比，AexPy 仅有 5 个，健壮性较强。

实用性上，在 45 个软件包的最新版本上，除去兼容和低破坏性变更，AexPy 检出了 433 个中或高破坏性变更，其中 405 个经人工检查真实存在。我们标记了 291 个记录在变更日志的变更，并向活跃包开发者报告了 63 个未记录的破坏性变更。我们

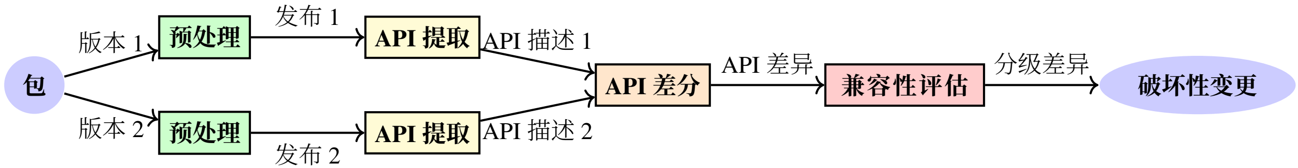

图 2 AexPy 破坏性变更检测分析工作流

收到了 50 个变更的回复，有 31 个被开发者确认。

效率上，我们在同一环境（具有 12 核 3.8GHz 主频 CPU 的 Ubuntu 18.04 主机上，每个版本对限制 50GB 内存和一小时分析时间）运行三个工具，来分析 45 个软件包的所有支持 Python 3.7 及以上的相邻版本对。在耗时最高的源码量为 10 万到 50 万行的软件包中，AexPy 、PyCompat、Pidiff 的平均耗时分别为 489.3 秒、115.8 秒、231.3 秒。三个工具分析耗时在同一数量级。

5 原型使用

为便于部署和使用，我们使用容器镜像封装了整个原型系统和相关依赖工具，并公开了浏览器访问接口和命令行接口。两种接口均支持完整系统功能，以方便开发人员阅读和使用检测结果，自动化工具分析和集成检测结果。命令行接口主要用于与自动化工具集成，相关文档可参见 AexPy 网站 [6]，这里主要介绍浏览器用户界面接口的使用。

我们基于 Miniconda3 构建 AexPy 镜像，推荐在安装有 Docker 的 Ubuntu 18.04 及以上的稳定发行版使用如下命令拉取 AexPy 镜像，并启动 AexPy 浏览器服务，绑定到主机的 8009 端口。通过浏览器访问系统前端网站（位于 `http://localhost:8009/report`），提交待分析包版本后，AexPy 在后端进行分析，完成后在前端网站展示分析报告和统计数据，如图 3所示。开发人员可以便捷直观地了解版本变化带来的破坏性变更情况，并查看详细的变更信息。

```
docker pull stardustdl/aexpy:main
docker run -d -p 8009:8008 \
    stardustdl/aexpy:main serve
```

系统也提供了对 AexPy 分析中间结果的访问和展示功能，如包源代码、API 信息、变更详情等，可通过前端网站的`/preprocess/`，`/extract/`，`/diff/`等子路径访问中间结果。开发人员可以通过这些信息了解 AexPy 工作流程，查看包中 API，或进行基于 AexPy 的二次开发。我们也提供了原型系统实例视频来展示完整系统功能[1]。

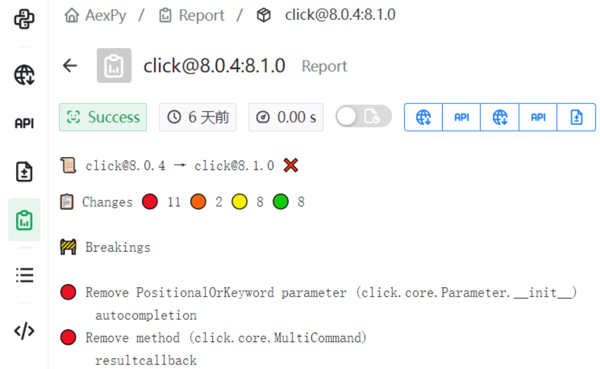

图 3 AexPy 前端网站分析报告页面

6 结论

本文介绍了 AexPy 原型系统，基于在 ISSRE’22 会议论文中提出的 Python 包 API 破坏性变更检测方法，聚焦健壮性、可靠性、扩展性、和效率进行系统设计实现。较现有工具，AexPy 以相近耗时水平，更全面地检测破坏性变更，检出的潜在变更得到了开发者确认，具有一定实用价值和推广前景。

参考文献

[1] DIG D, JOHNSON R E. How do apis evolve? A story of refactoring [J/OL]. J. Softw. Maintenance Res. Pract., 2006, 18(2):83-107. DOI: 10.1002/smr.328.

[2] DU X, MA J. Aexpy: Detecting api breaking changes in python packages [C]//33rd IEEE International Symposium on Software Reliability Engineering, ISSRE 2022. [S.l.]: IEEE, 2022: (forthcoming).

[3] The Mypy Project. mypy - optional static typing for python[EB/OL]. 2022. http://www.mypy-lang.org/.

[4] REDDY M. Api design for c++[M]. [S.l.]: Elsevier, 2011.

[5] ZHANG Z, ZHU H, WEN M, et al. How do python framework apis evolve? an exploratory study[C/OL]//27th IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2020. IEEE, 2020: 81-92. DOI: 10.1109/SANER48275.2020.9054800.

[6] DU X. Home - aexpy[EB/OL]. 2022. https://aexpy.netlify.app/.

[7] MCGOVERN R. rohanpm/pidiff: The python interface diff tool[EB/OL]. 2022. https://github.com/rohanpm/pidiff.

[1]AexPy 演示视频，https://www.bilibili.com/video/BV1PG411F77m/